# Web报表服务平台-项目搭建

## 选用技术

* Spring、SpringMVC、Mybatis
* JSP、JSTL、jQuery、jQuery plugin、EasyUI、Bootstrap、CSS+DIV
* Redis（缓存服务器）
* httpclient（调用系统服务）
* Mysql
* Nginx（web服务器）

## 开发工具和环境

Eclipse(jee)

apache-maven-3.5.2（开发工具自带）

Tomcat 7.0.52（Maven Tomcat Plugin）

**JDK 1.8**

Mysql 5.6

Nginx 1.8.0

Redis 3.0.0

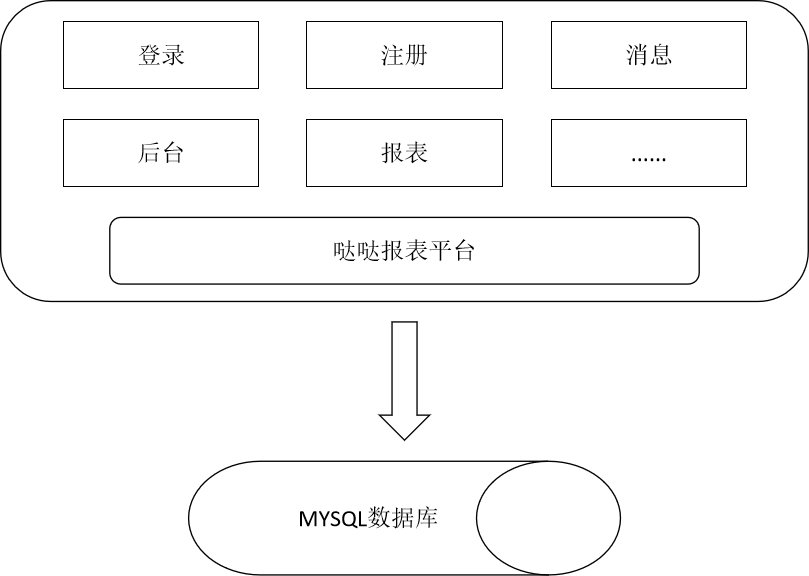
Win8 操作系统

**Git（版本管理）**

## 后台管理系统工程结构

**系统架构：传统架构、分布式架构**

**传统架构：**



传统架构

**存在问题：**

模块之间耦合度太高，其中一个升级其他都得升级

开发困难，各个团队开发最后都要整合一起

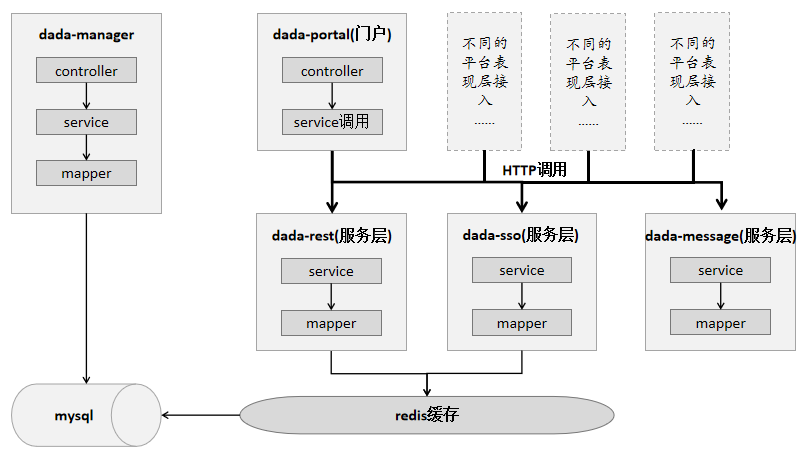
系统的扩展性差

不能灵活的进行分布式部署

**解决方法：**

把模块拆分成独立的工程，单点运行。如果某一个点压力大可以对这一个点单独增加配置。其他的点不受影响。但系统之间交互需要额外的工作量来进行接口的开发。

**分布式**：把系统拆分成多个工程，要完成系统的工程需要多个工程协作完成



分布式架构

把系统按照模块拆分成多个子系统

优点：

把模块拆分，使用接口通信，降低模块之间的耦合度

把项目拆分成若干个子项目，不同的团队负责不同的子项目

增加功能时只需要再增加一个子项目，调用其他系统的接口就可以

可以灵活的进行分布式部署

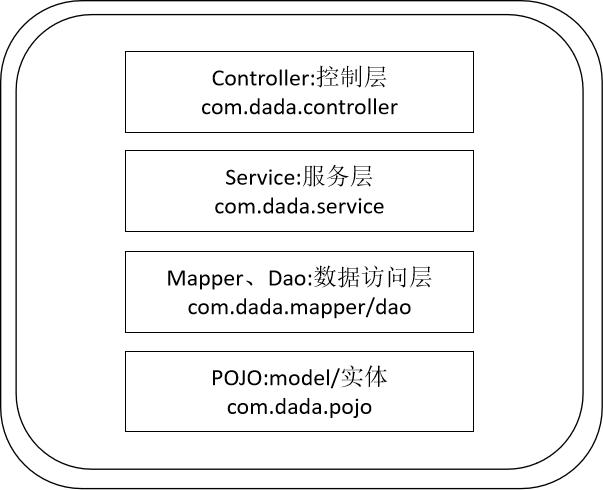
缺点：

系统之间交互需要使用远程通信，接口开发增加工作量

## 后台管理系统工程搭建

工程结构说明（传统工程结构、maven管理的工程结构）

传统工程结构：



Maven管理的工程结构：

不使用maven：工程部署时需要手动复制jar包，完成工程构建，过程非常繁琐;

使用maven进行工程构建：使用maven可以实现一步构建

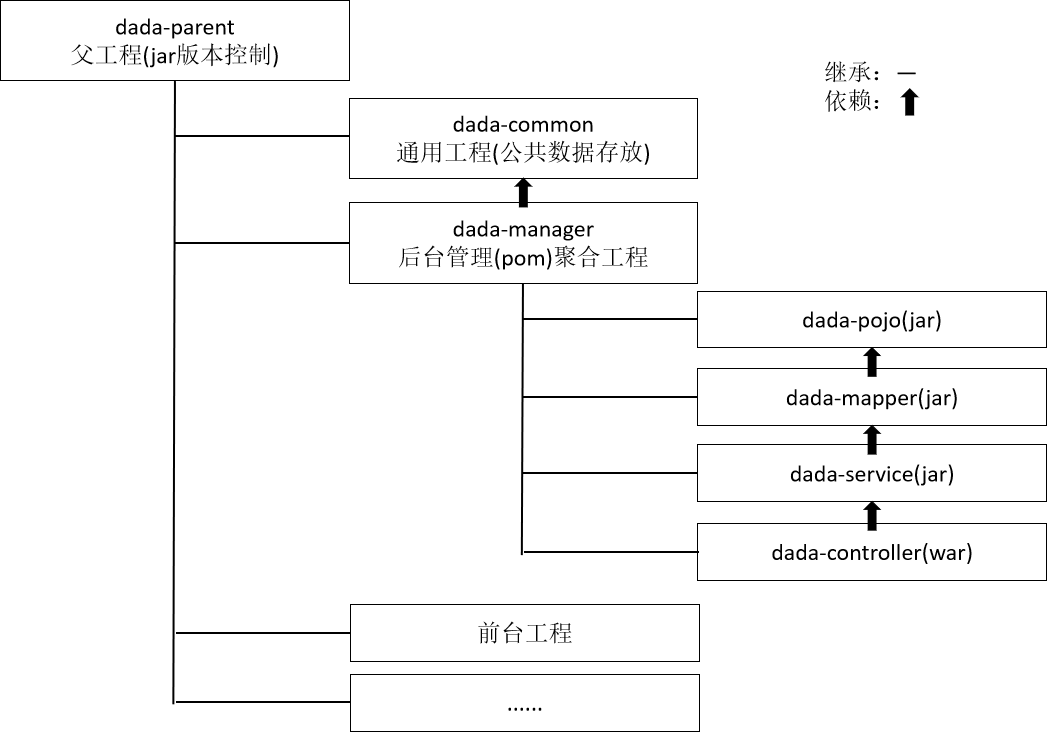
Maven：

依赖管理、jar包、工程之间的依赖;

项目构建，实现项目的一步构建;

工程聚合、继承、依赖;

Maven的工程类型包括：war包工程、Jar包工程、Pom工程



后台管理系统工程结构：

dada-parent -- 管理依赖jar包的版本，全局，公司级别

|--dada-common --- 通用组件、工具类

|--dada-manage -- 后台系统

|--com.dada.manage.web

|--com.dada.manage.service

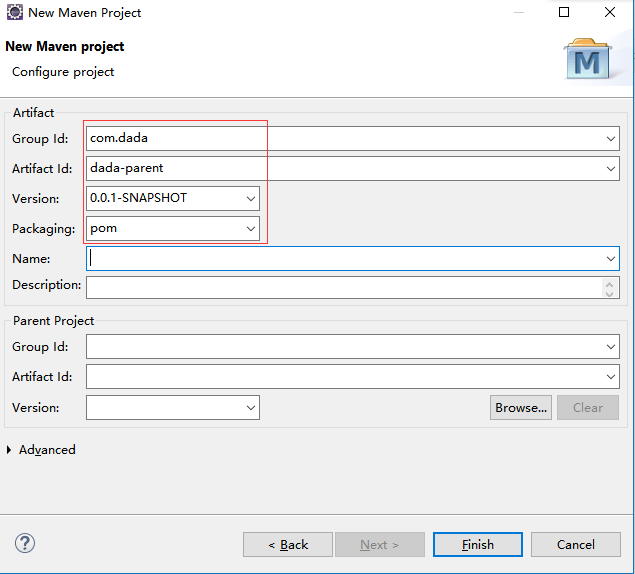
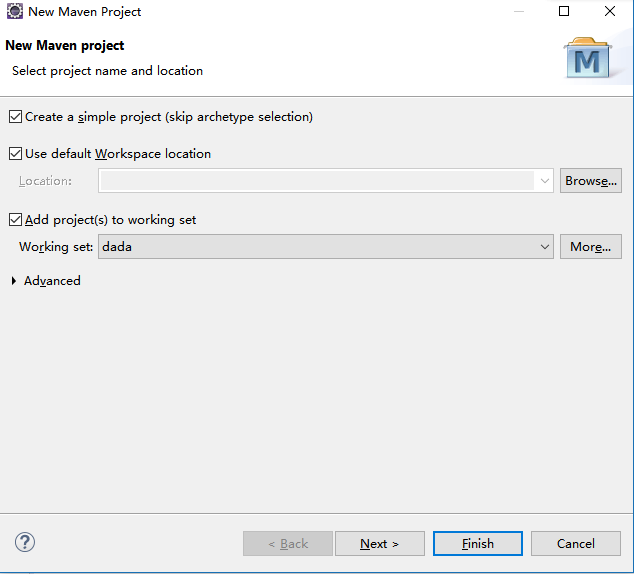
|--com.dada.manage.mapper

|--com.dada.manage.pojo

### 父工程dada-parent的搭建

父工程为一个pom工程，在父工程中定义依赖的jar包的版本信息、Maven插件的版本

#### 1.创建maven工程



#### 2.修改pom.xml文件

分页插件待修正！

|  |
| --- |
|  |

#### 3.将dada-parent安装到本地仓库

右键dada-parent工程，选择Run As --> maven install

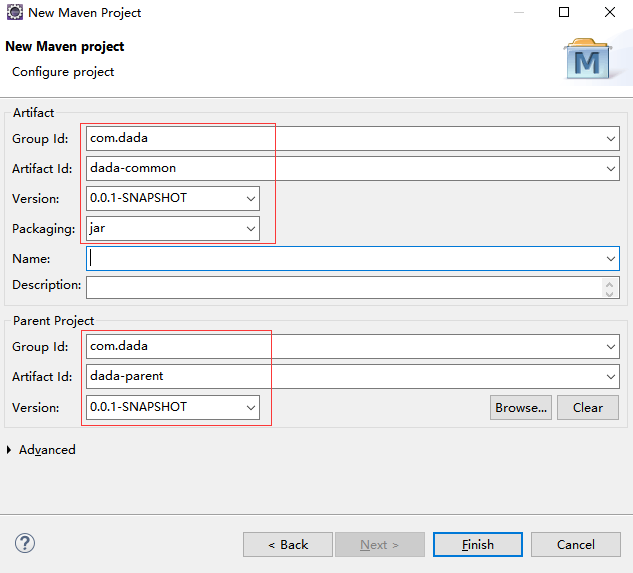
或者是直接进入后台Terminal，进入到相应的工程目录下，执行maven指令：mvn clean install

指令执行完成，可以看到当前工程被加载到本地仓库对应的路径下

### dada-common的搭建

#### 1.创建maven工程

Common工程需要继承taotao-parent工程



#### 2.修改pom.xml文件

修改taotao-common工程的pom文件，在文件中添加对taotao-parent的继承

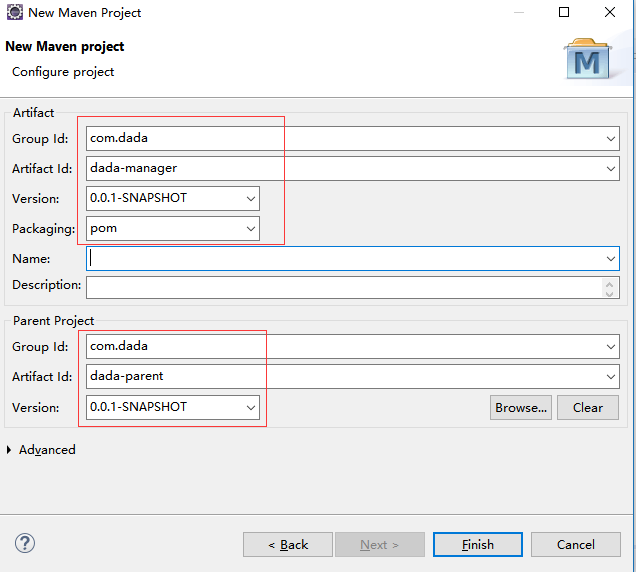
|  |
| --- |
|  |

#### 3.更新工程

右键dada-common更新工程，maven-->update Project

### dada-manager的搭建

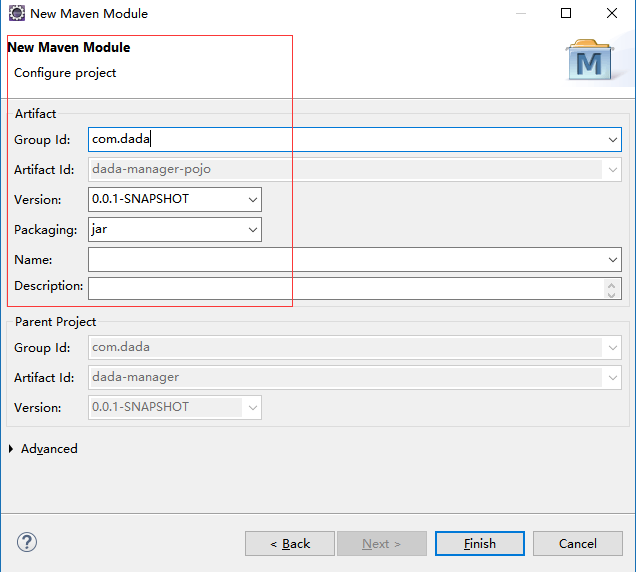
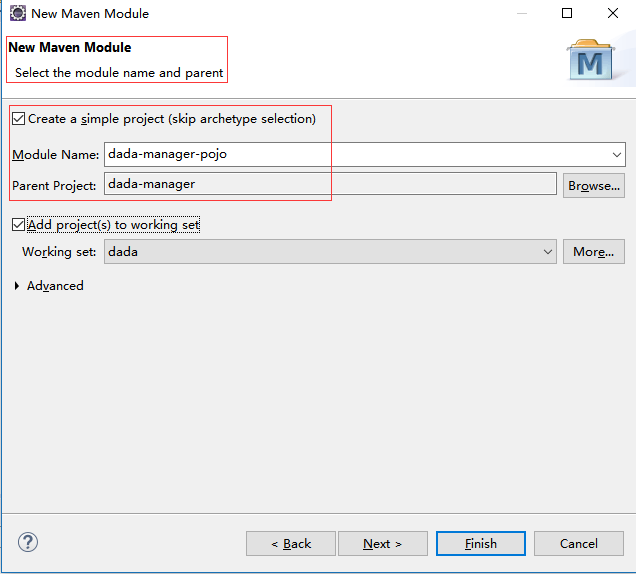
#### 1.创建dada-manager



修改pom.xml文件：

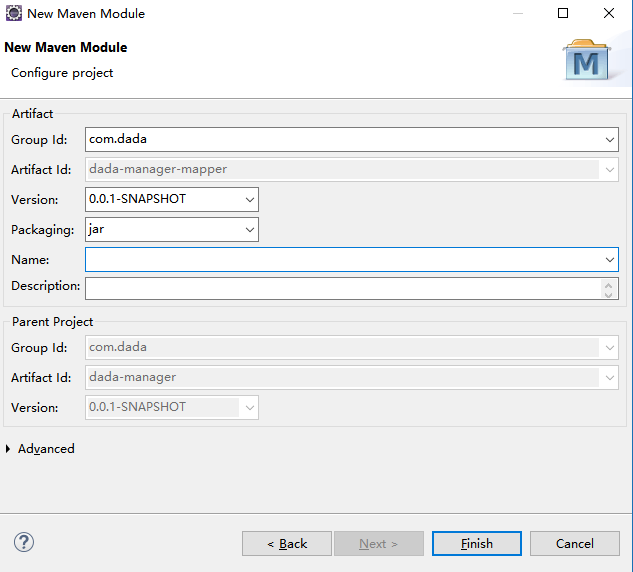
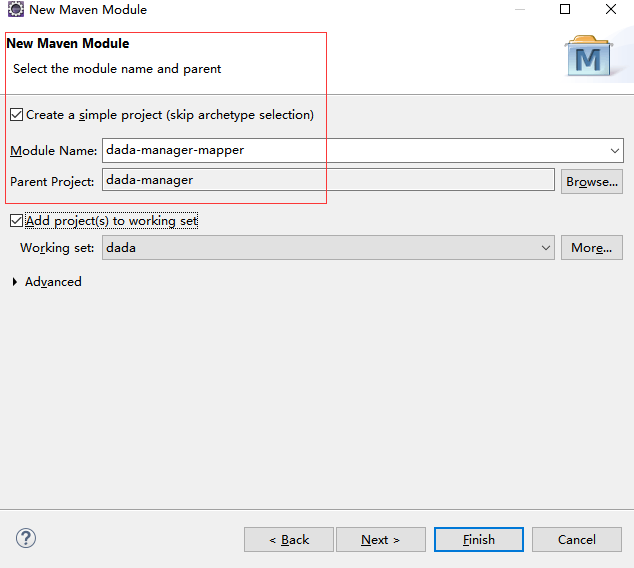
<project xmlns=*"http://maven.apache.org/POM/4.0.0"*

#### 2.创建dada-manager-pojo



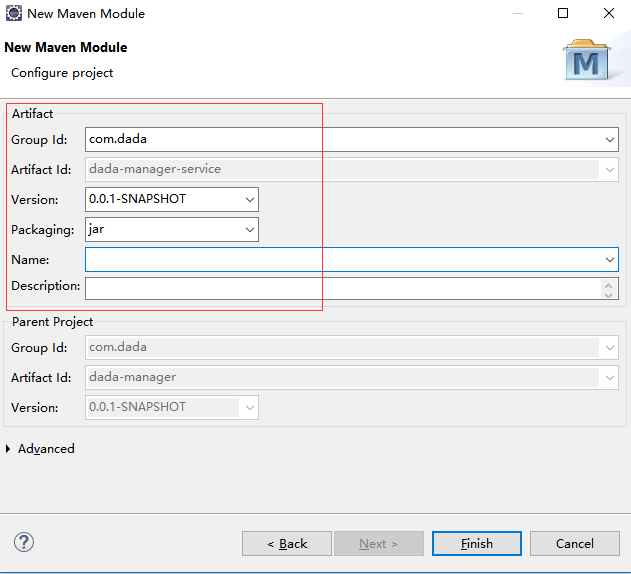
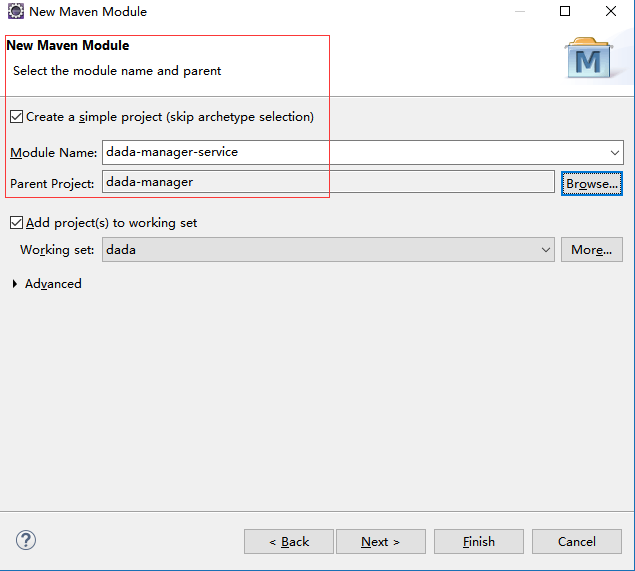
taotao-manager-pojo不需要依赖任何jar报，因此此处不做修改

#### 3.创建dada-manager-mapper



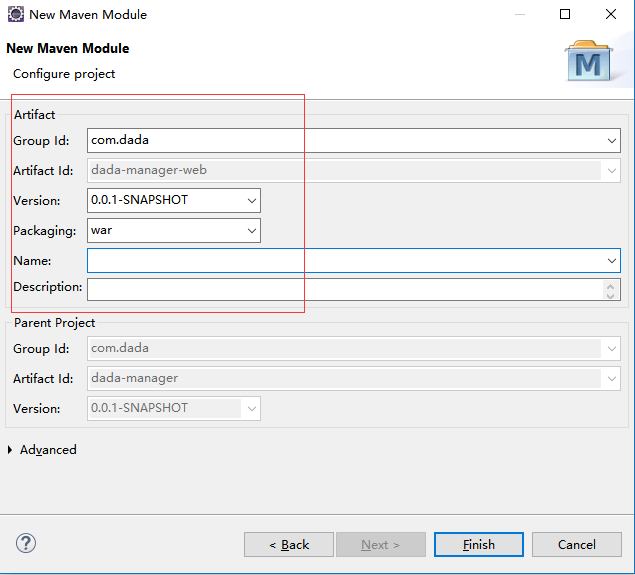
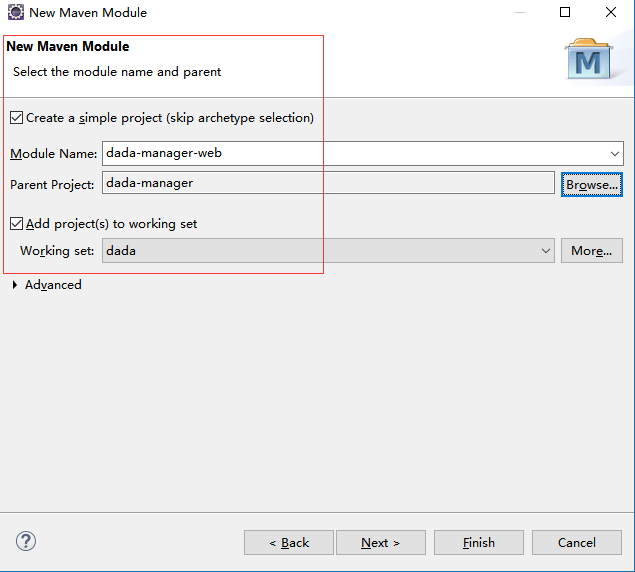
pom.xml文件修改：

#### 4.创建dada-manager-service



pom.xml文件修改:

#### 5.创建dada-manager-web

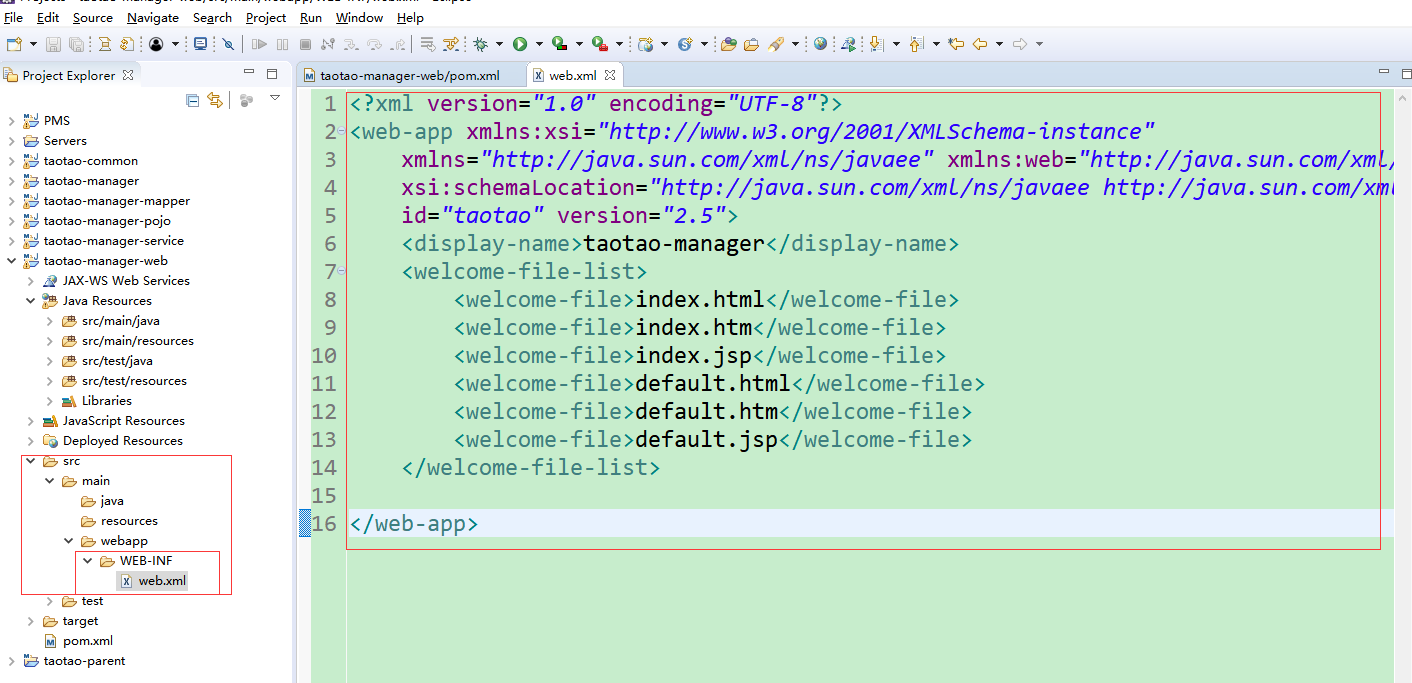


##### pom.xml文件配置：

修改pom.xml文件：

##### web.xml配置

进入taotao-manager-web下，在相应目录创建一个WEB-INF文件夹，在其下创建一个web.xml



|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://java.sun.com/xml/ns/javaee"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  id=*"taotao"* version=*"2.5"*>  <display-name>taotao-manager</display-name>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  <welcome-file>index.htm</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  <welcome-file>default.htm</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>  </web-app> |

在相应webapp下创建一个index.jsp文件进行测试

#### 6.配置tomcat插件

运行web工程需要添加一个tomcat插件。插件必须添加到taotao-manager工程中。因为taotao-manager是聚合工程。在运行时需要把子工程聚合到一起才能运行。

taotao-manager中加载配置tomcat插件，其与taotao-parent中加载的插件对应一致

|  |
| --- |
| <build>  <plugins>  <!-- 配置Tomcat插件 -->  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.2</version>  <configuration>  <port>8080</port>  <path>/</path>  </configuration>  </plugin>  </plugins>  </build> |

启动tomcat命令：tomcat7:run，是maven自带的tomcat插件，其使用如下

右键taotao-manager，Run As-->maven builder(第2个)，执行指令“clean tomcat7:run”

使用maven命令：

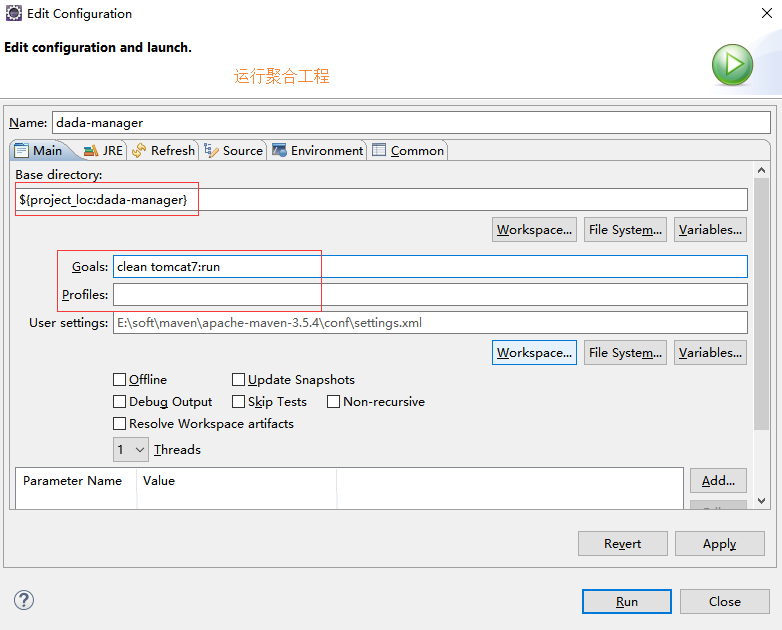
clean tomcat7:run

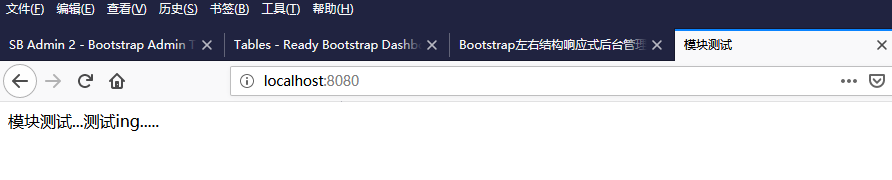
tomcat7指定使用tomcat7的插件。

注意：执行mvn clean install

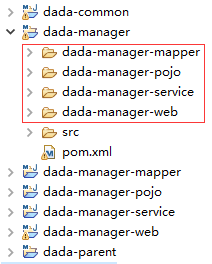
需要把taotao-parent工程安装到本地仓库

需要把taotao-common安装到本地仓库





### dada-manage子模块依赖关系



依赖关系：

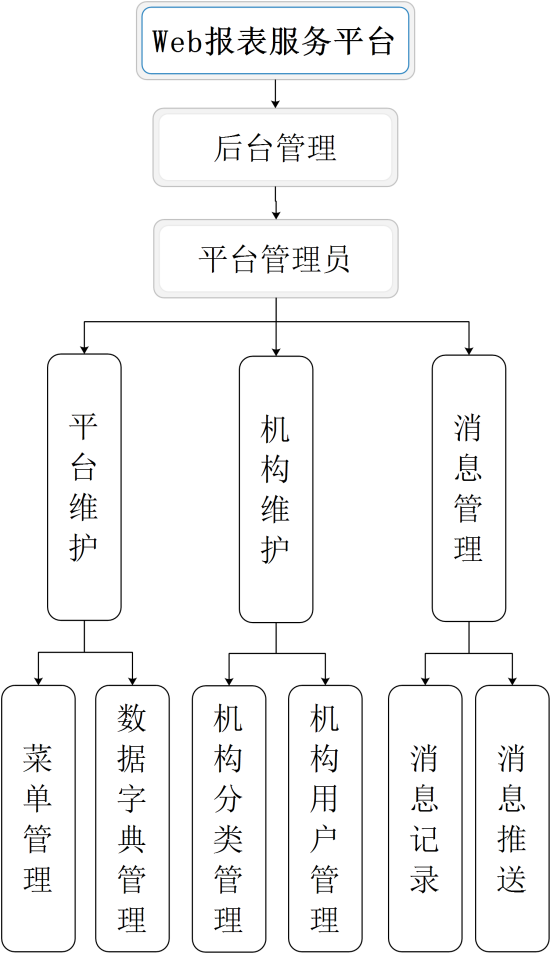
web 🡺 service

service 🡺 mapper

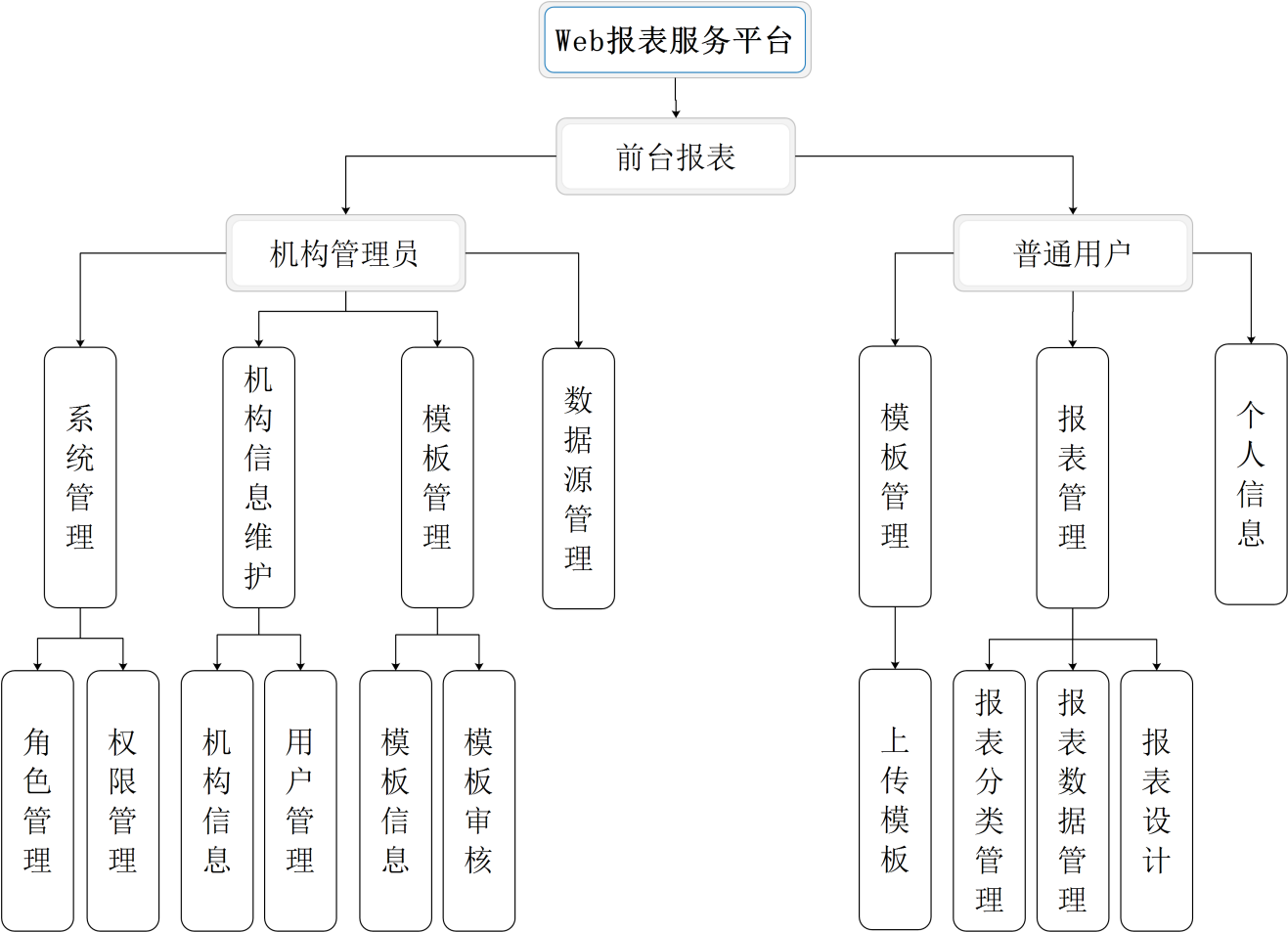
mapper 🡺 pojo

# SSM框架整合

## 报表平台设计



**后台管理-系统服务构图**



**前台报表-系统服务构图**

基本内容：

1.创建dada-manager数据库，引入数据库信息

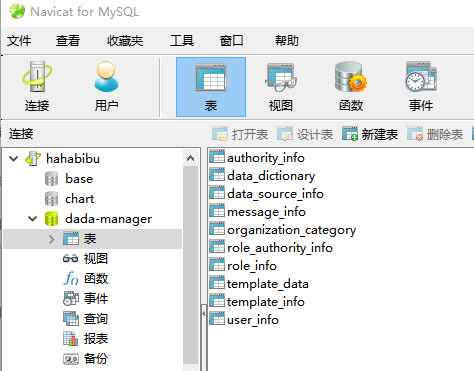
2.使用mybatis逆向工程生成相应的pojo、dao（mapper）

3.分别整合dao、service、controller层

4.整合项目测试分析

## 1.创建dada数据库，引入数据库信息

引入dada-manager.sql,查看每个表的关联关系



## 2.使用mybatis逆向工程生成相应的pojo、dao（mapper）

在使用mybatis逆向工程生成代码时，如果是已经生成的内容需要重新生成，则需要将原有的代码内容删除，否则第二次生成时其并不会在原有的文件进行覆盖，而是在原有的文件基础上追加内容，但并不报错，如此导致项目执行失败。

## 3.分别整合dao、service、controller层

### SSM框架整合

#### <1>整合的思路

##### Dao层

使用mybatis框架。创建SqlMapConfig.xml。

创建一个applicationContext-dao.xml

1. 配置数据源
2. 需要让spring容器管理SqlsessionFactory，单例存在。
3. 把mapper的代理对象放到spring容器中。使用扫描包的方式加载mapper的代理对象。

##### Service层

1. 事务管理
2. 需要把service实现类对象放到spring容器中管理。

##### Controller层(表现层)

1. 配置注解驱动
2. 配置视图解析器
3. 需要扫描controller

##### Web.xml

1. spring容器的配置
2. Springmvc前端控制器的配置
3. Post乱码过滤器

**考虑配置文件存放的位置，则需要对相关的工程结构进行分析，pom、jar类型的maven工程中存放配置文件没有办法进行读取，因此考虑需要把配置文件放到taotao-manager-web工程下，因为此工程为war工程。**

**也可理解成项目运行时配置文件的读取时由tomcat进行操作的，此举为了迎合tomcat的需求？**

**换个角度思考，如果将配置文件放在某个jar类型的maven工程中，其他项目在依赖的时候是通过lib引入该jar包，因此无法访问其内部的配置文件**

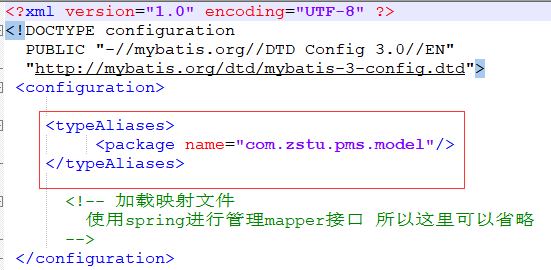
#### <2>整合框架

在taotao-manager-web下对应src/main/resource下创建3个文件夹分别存放相应的配置文件，分别为“mybatis”（存放mybatis整合相关的配置文件）、“resource”（存放基本的配置文件，例如数据库连接文件db.properties）、“spring”（存放spring整合相关的配置文件）

##### Mybatis整合

在mybatis文件夹下添加“SqlMapConfig.xml”文件（可在该文件中设置mapper配置相关的内容）

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <configuration>    </configuration> |



在spring文件夹下创建applicationContext-dao.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- 数据库连接池 -->  <!-- 加载配置文件 -->  <context:property-placeholder location=*"classpath:resource/db.properties"* />  <!-- 数据库连接池 -->  <bean id=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"*  destroy-method=*"close"*>  <property name=*"url"* value=*"${jdbc.url}"* />  <property name=*"username"* value=*"${jdbc.username}"* />  <property name=*"password"* value=*"${jdbc.password}"* />  <property name=*"driverClassName"* value=*"${jdbc.driver}"* />  <property name=*"maxActive"* value=*"10"* />  <property name=*"minIdle"* value=*"5"* />  </bean>  <!-- 配置sqlsessionFactory -->  <bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <property name=*"configLocation"* value=*"classpath:mybatis/SqlMapConfig.xml"*></property>  <property name=*"dataSource"* ref=*"dataSource"*></property>  </bean>  <!-- 配置扫描包，加载mapper代理对象 -->  <bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <property name=*"basePackage"* value=*"com.taotao.mapper"*></property>  </bean>  </beans> |

##### Service层整合

在spring文件夹下创建applicationContext-service.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- 扫描包加载Service实现类 -->  <context:component-scan base-package=*"com.taotao.service"*></context:component-scan>  </beans> |

在spring文件夹下创建applicationContext-transaction.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- 事务管理器 -->  <bean id=*"transactionManager"*  class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <!-- 数据源 -->  <property name=*"dataSource"* ref=*"dataSource"* />  </bean>  <!-- 通知 -->  <tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>  <tx:attributes>  <!-- 传播行为 -->  <tx:method name=*"save\*"* propagation=*"REQUIRED"* />  <tx:method name=*"insert\*"* propagation=*"REQUIRED"* />  <tx:method name=*"add\*"* propagation=*"REQUIRED"* />  <tx:method name=*"create\*"* propagation=*"REQUIRED"* />  <tx:method name=*"delete\*"* propagation=*"REQUIRED"* />  <tx:method name=*"update\*"* propagation=*"REQUIRED"* />  <tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"select\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  </tx:attributes>  </tx:advice>  <!-- 切面 -->  <aop:config>  <aop:advisor advice-ref=*"txAdvice"*  pointcut=*"execution(\* com.taotao.service.\*.\*(..))"* />  </aop:config>  </beans> |

##### Controller层整合

在service文件夹中添加springmvc.xml文件

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:mvc=*"http://www.springframework.org/schema/mvc"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>  <!-- 配置包扫描器 -->  <context:component-scan base-package=*"com.taotao.controller"* />  <!-- 配置注解驱动 -->  <mvc:annotation-driven />  <!-- 配置视图解析器 -->  <bean  class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"prefix"* value=*"/WEB-INF/jsp/"* />  <property name=*"suffix"* value=*".jsp"* />  </bean>  <!-- 配置资源映射(针对静态资源,防止被拦截) -->  <mvc:resources location=*"/WEB-INF/css/"* mapping=*"/css/\*\*"*/>  <mvc:resources location=*"/WEB-INF/js/"* mapping=*"/js/\*\*"*/>  </beans> |

##### Web.xml

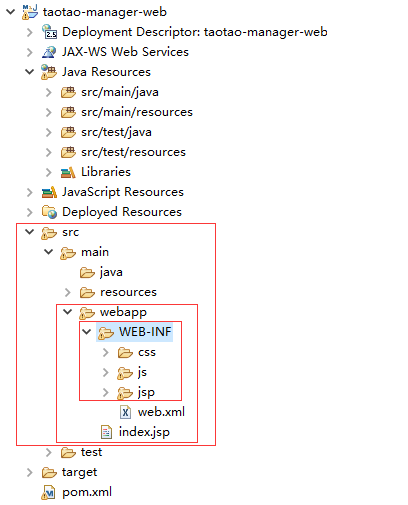
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://java.sun.com/xml/ns/javaee"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  id=*"taotao"* version=*"2.5"*>  <display-name>taotao-manager</display-name>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  <welcome-file>index.htm</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  <welcome-file>default.htm</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>  <!-- 加载spring容器 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/applicationContext-\*.xml</param-value>  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <!-- 解决post乱码 -->  <filter>  <filter-name>CharacterEncodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>utf-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>CharacterEncodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  <!-- springmvc的前端控制器 -->  <servlet>  <servlet-name>taotao-manager</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <!-- contextConfigLocation不是必须的， 如果不配置contextConfigLocation， springmvc的配置文件默认在：WEB-INF/servlet的name+"-servlet.xml" -->  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/springmvc.xml</param-value>  </init-param>  <!-- load-on-startup元素标记容器是否应该在web应用程序启动的时候就加载这个servlet -->  <load-on-startup>1</load-on-startup>  </servlet>  <!-- 配置拦截url的请求:/表示拦截所有的请求(包括静态资源) -->  <servlet-mapping>  <servlet-name>dada-manager</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  </web-app> |

/：会拦截所有请求包括静态资源。需要在springmvc.xml中添加静态资源的映射，否则无法直接访问。

|  |
| --- |
| <!-- 配置资源映射 -->  <mvc:resources location=*"/WEB-INF/css/"* mapping=*"/css/\*\*"*/>  <mvc:resources location=*"/WEB-INF/js/"* mapping=*"/js/\*\*"*/> |

#### <3>添加静态资源

将相关的静态文件资料添加到taotao-manager-web工程的对应的src/main/webapp/WEB-INF文件夹下



#### <4>Springmvc和spring的父子容器关系



问题：**为什么不使用全局扫描？**例如在service扫描的时候直接配置com.taotao，理论上实现service、controller及相关所有内容的扫描。

例如：

在applicationContext-service中配置全局扫描：

<!-- 扫描包加载Service实现类 -->

<context:component-scan base-package=*"com.taotao"*></context:component-scan>

会扫描@Controller、@Service、@Repository、@Compnent

Springmvc.xml中不配置扫描：

结论：springmvc不能提供服务，因为springmvc子容器中没有controller对象。因此如此配置会导致在访问数据的时候报404错误！

可以在springmvc.sml中配置如下：

<!-- 定义扫描包的路径 -->

<context:component-scan base-package="com.taotao.controller,com.taotao.service.impl"></context:component-scan>

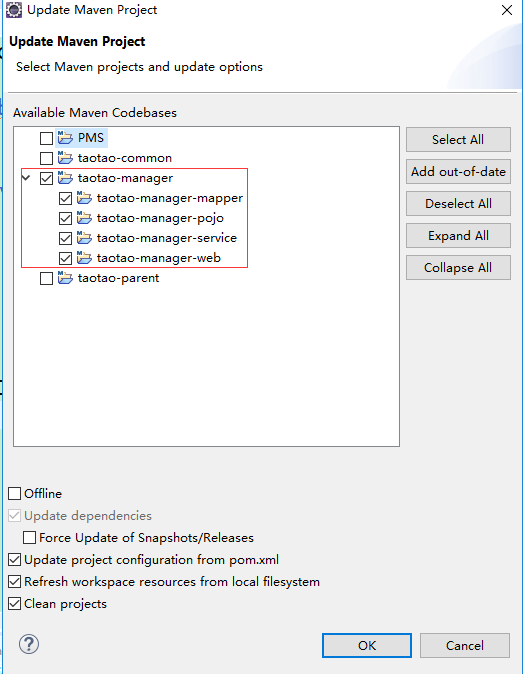
在spring+springMVC架构下，如果要将事务配置到Controller层是无法实现的（事务管理器无法访问controller），必须把事务配置到service层。

#### <5>测试整合结果

需求分析：跟据商品id查询商品信息（商品id为536563）

Sql语句：SELECT \* from tb\_item WHERE id=536563

在对应的工程添加相应的dao、service、controller内容，添加完成选择Maven更新主工程进行测试（taotao-manager-dao对应dao层、taotao-manager-service/impl对应service层、taotao-manager-web对应controller层）



Dao层：可以使用逆向工程生成的mapper文件

Service层：接收商品id调用dao查询商品信息。返回商品pojo对象

|  |
| --- |
|  |

Controller层：接收页面请求商品id，调用service查询商品信息。直接返回一个json数据。需要使用@ResponseBody注解。

|  |
| --- |
|  |

##### 整合测试时遇到的问题：

###### 问题1：tomcat启动

在启动tomcat报错，由于maven依赖没有解决，除了taotao-parent、taotao-common之外，还需要依次将依赖的工程taotao-manager、taotao-manager-pojo、taotao-manager-mapper、taotao-manager-service依次加载到本地仓库中（被taotao-manager-web依赖调用），随后再在taotao-manager-web下执行mvn clean install进行测试。

**即运行聚合工程时必须先将依赖相关的工程install到本地仓库**



解决思路：进入到taotao-manager所在目录，执行mvn clean install指令查看问题

如果重复启动出现如下问题：

[ERROR] Failed to execute goal org.apache.maven.plugins:maven-clean-plugin:2.5:clean (default-clean) on project taotao-rest: Failed to clean project: Failed to delete C:\Users\eclipsework-space\taotao-rest\target\tomcat\logs\access\_log.2017-08-14 -> [Help 1]

[ERROR]

[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.

[ERROR] Re-run Maven using the -X switch to enable full debug logging.

[ERROR]

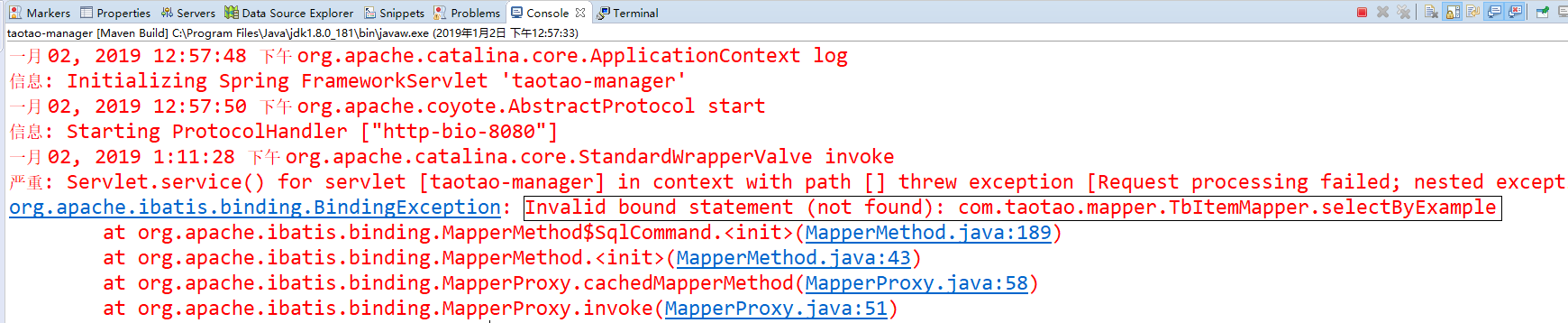
[ERROR] For more information about the errors and possible solutions, please read the following articles:

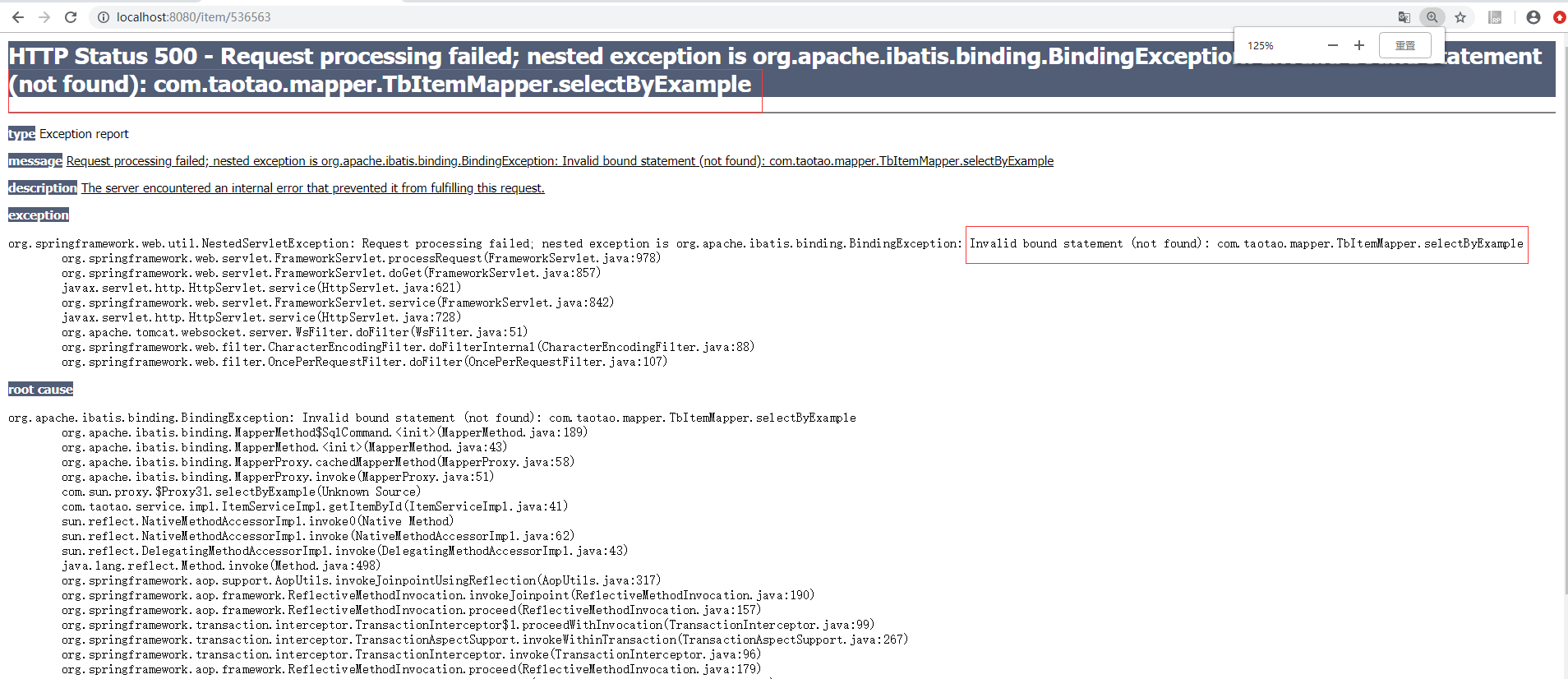
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoExecutionException

解决思路：原因是maven工程重复操作，资源被占用导致部分数据无法删除，第二次运行项目的时候记得一定要把前一次运行的状态关掉，再重新运行，不然就会报这个错误

###### 问题2：访问controller报500错误

找不到相应的mapper映射文件：





解决思路：

没有mapper映射文件：查看对应工作路径下的taotao-manager-mapper下target/classess/...对应文件夹查看是否生成xxxExample.class

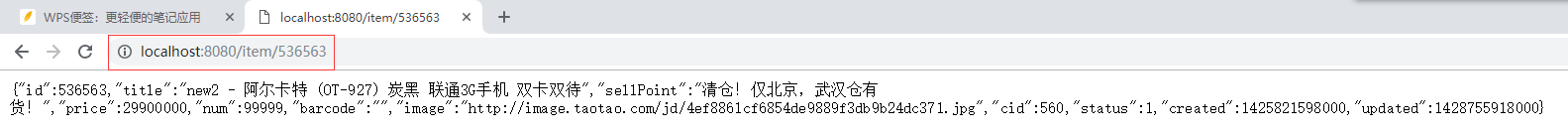
参考路径：工作目录\taotao-manager\taotao-manager-mapper\target\classes\com\taotao\mapper

解决方法：

默认是不会编译该类型文件的，因此需要制定编译的信息，修改taotao-manager-mapper工程的pom文件，在pom文件中添加如下内容：

|  |
| --- |
| <!-- 如果不添加此节点mybatis的mapper.xml文件都会被漏掉。 -->  <build>  <resources>  <resource>  <directory>src/main/java</directory>  <includes>  <include>\*\*/\*.properties</include>  <include>\*\*/\*.xml</include>  </includes>  <filtering>false</filtering>  </resource>  </resources>  </build> |

重新启动，再次测试OK：



出现以上界面，说明框架搭建完成

###### 问题3：资源文件访问404

Web.xml文件中配置了拦截的内容，如果是“/”默认会拦截所有的请求，包括相关的静态资源，导致相关的css、js文件被拦截而失效，需要调整相应的数据

<servlet-mapping>

<servlet-name>MyServlet</servlet-name>

<url-pattern>xxxxxx</url-pattern>

<url-pattern>yyyyyyy</url-pattern>

</servlet-mapping>

<servlet>标签首先配置声明一个servlet，包括servlet的名字和对应的java类名。  
其中<servlet-mapping>标签声明了与该servlet相应的匹配规则，每个<url-pattern>标签代表1个匹配规则。

在配置了Servlet与URL样式之间的映射后，当Servlet容器接收到一个请求，它首先确定该请求应该由哪一个Web应用程序来响应。这是通过比较请求URI的开始部分与Web应用程序的上下文路径来确定的。映射到Servlet的路径是请求URI减去上下文的路径，Web应用程序的 Context对象在去掉请求URI的上下文路径后，将按照下面的路径映射规则的顺序对剩余部分的路径进行处理，并且在找到第一个成功的匹配后，不再进行 下一个匹配。

容器试着对请求的路径和Servlet映射的路径进行精确匹配，如果匹配成功，则调用这个Servlet来处理请求。

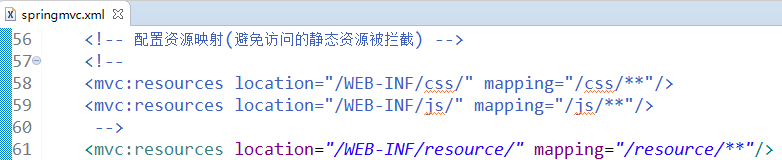
容器试着匹配最长的路径前缀，以斜杠（/）为路径分隔符，按照路径树逐级递减匹配，选择最长匹配的Servlet来处理请求。

如果请求的URL路径最后有扩展名，如.jsp，Servlet容器[会试](https://www.baidu.com/s?wd=%E4%BC%9A%E8%AF%95&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/Star_CSU/article/details/_blank)着匹配处理这个扩展名的Servlet。

如果按照前面3条规则没有找到匹配的Servlet，容器会调用Web应用程序默认的Servlet来对请求进行处理，如果没有定义默认的Servlet，容器将向客户端发送HTTP 404错误信息（请求资源不存在）。

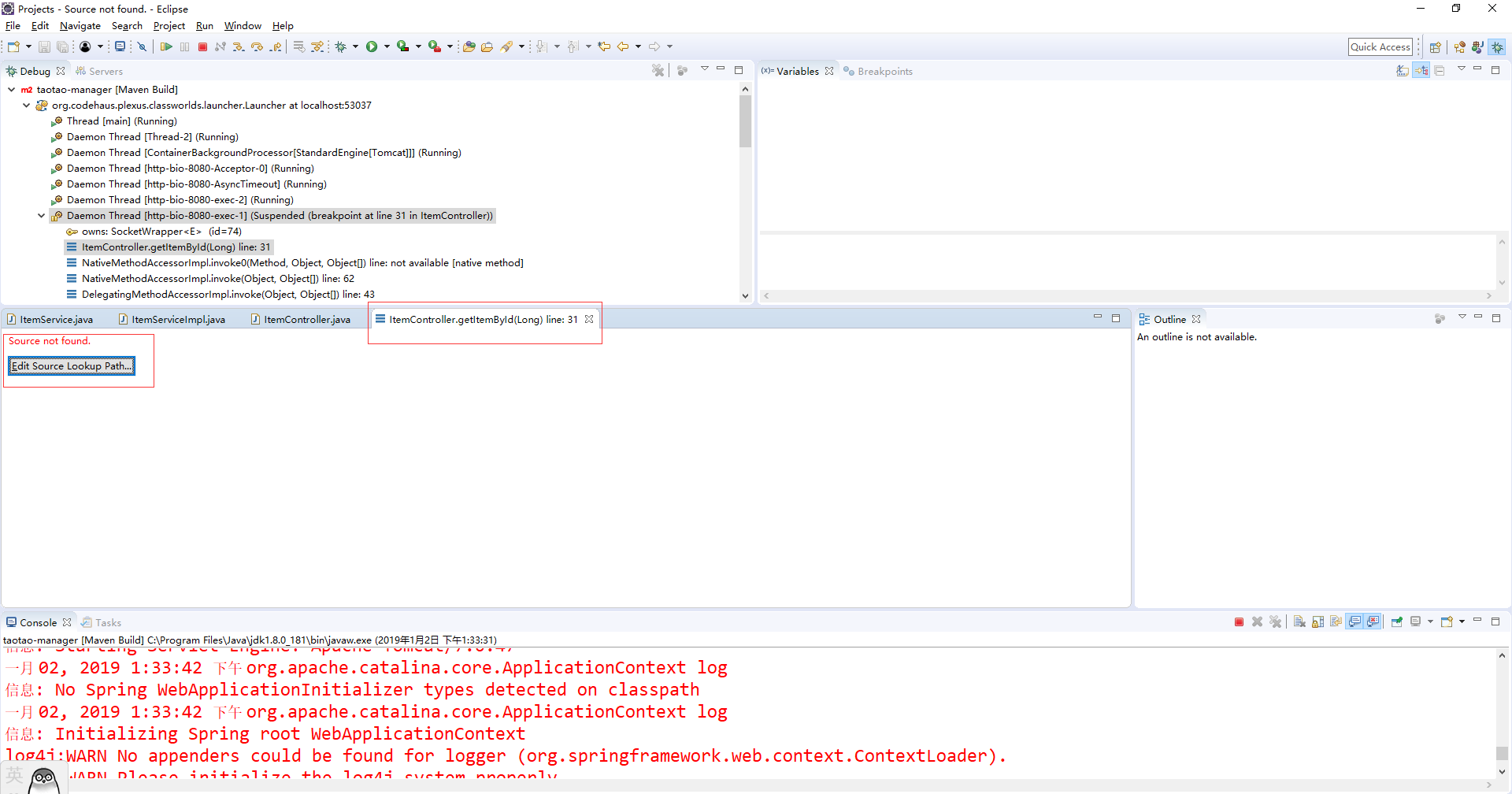


调整：**在springmvc.xml中配置相应的资源映射即可**

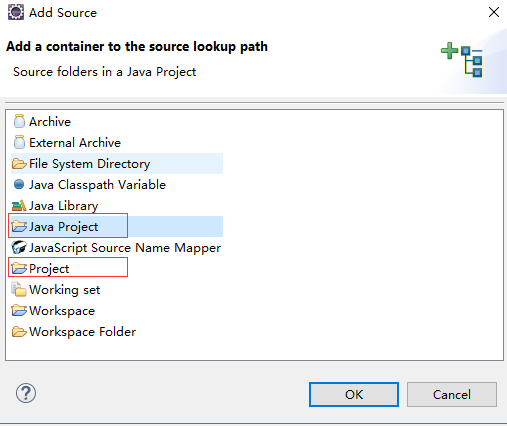
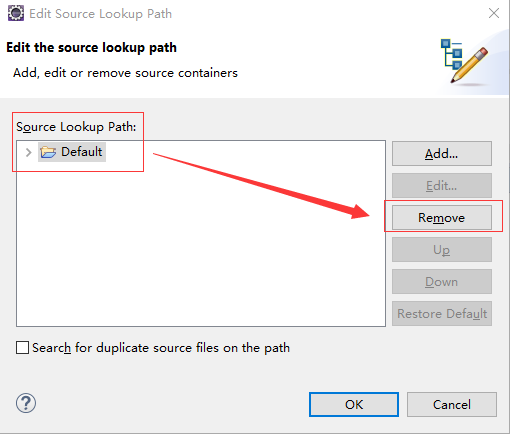


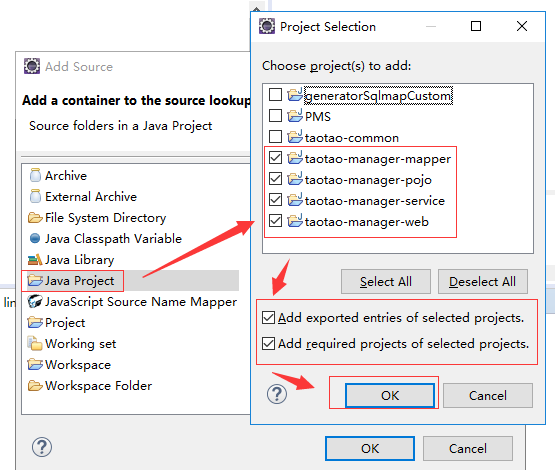
#### <6>使用maven的tomcat插件时debug

弹出提示框点击yes进入debug模式：

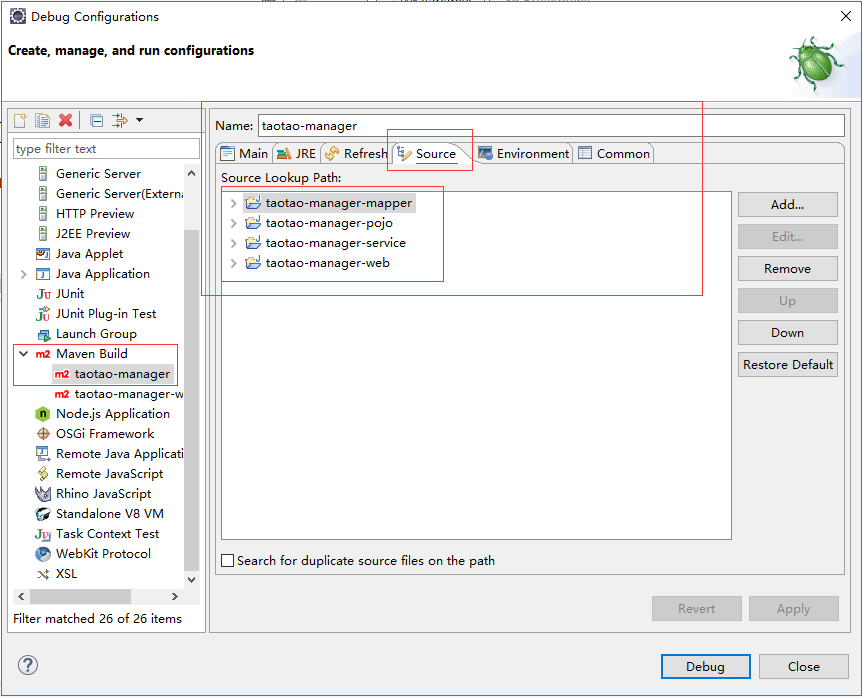


点击“edit Source Lookup Path...”，移除默认的任荣，添加指定的project，此处最好选择“Java Project”（Project所有的工程列出，不便筛选，且有些工程是pom属性），配置完成下次启动生效





也可在debug调试之前进行配置，debug下拉项有个“debug configuration”，进入相应工程的配置项，此处为taotao-manager，在source选项卡下配置相关内容

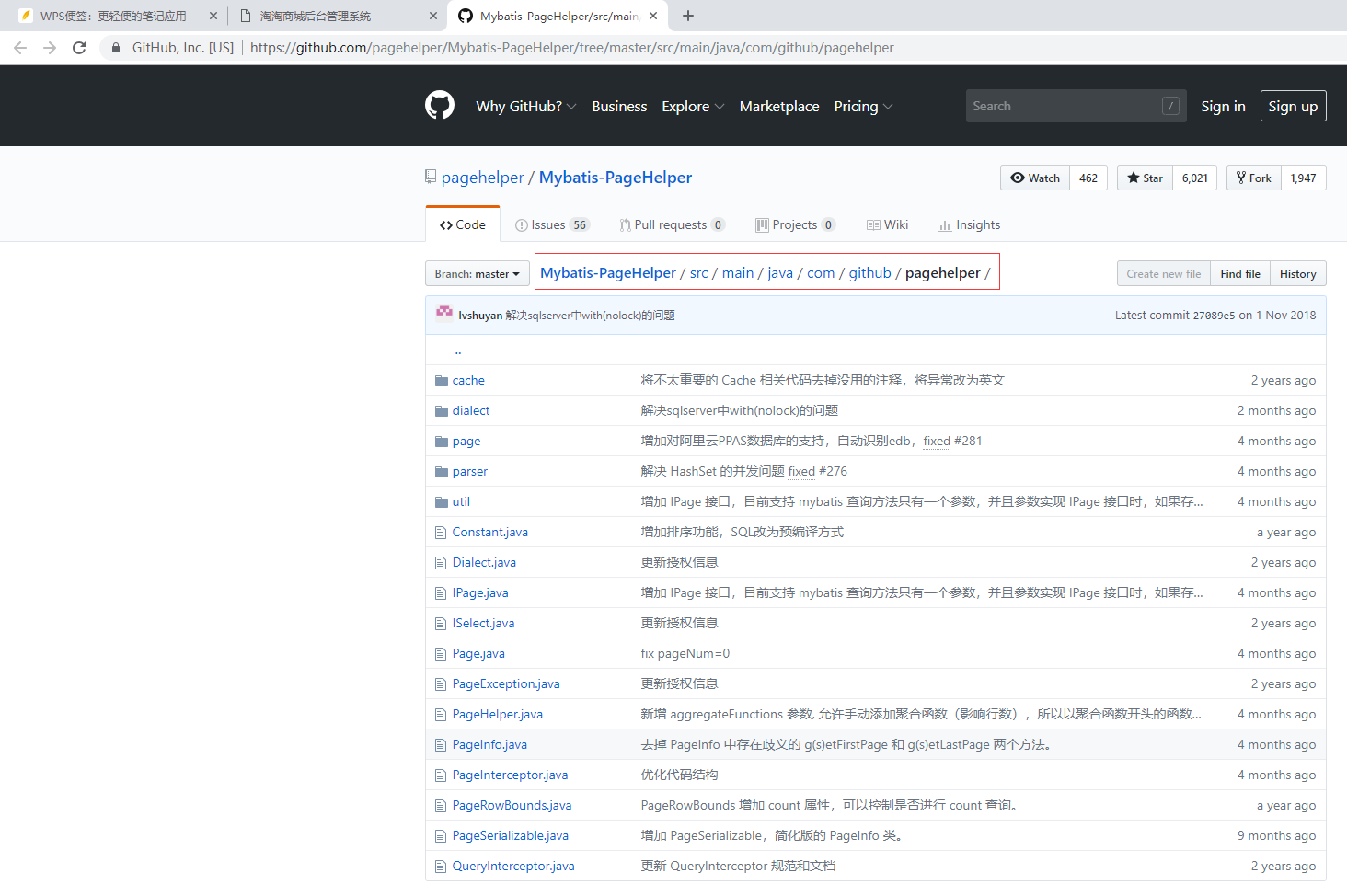


# 附加说明

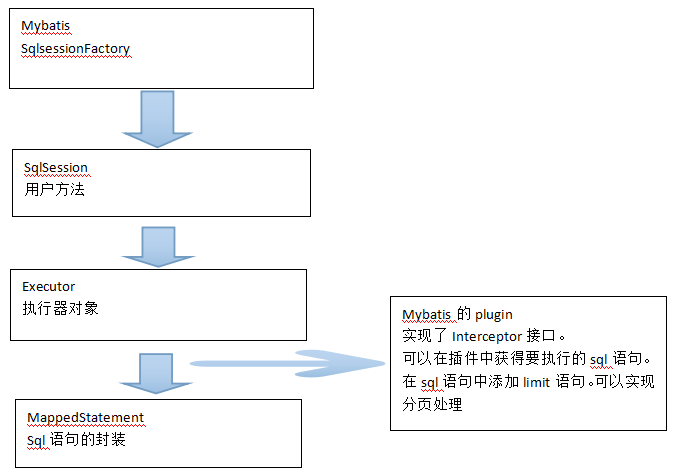
## 分页插件PageHelper

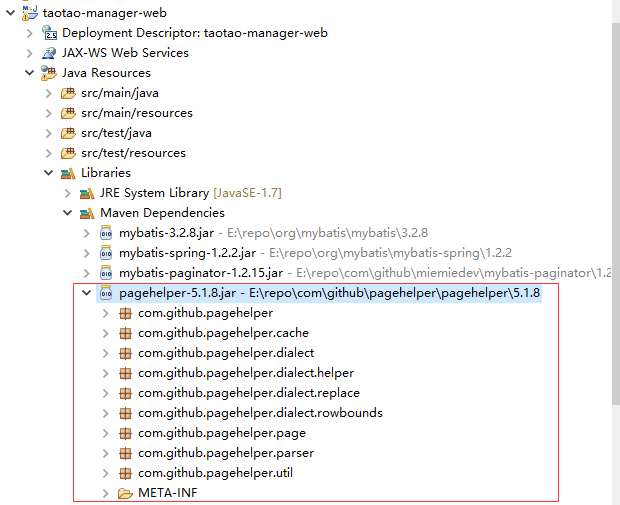
## 官方网站：

<https://github.com/pagehelper/Mybatis-PageHelper/tree/master/src/main/java/com/github/pagehelper>



## 实现原理





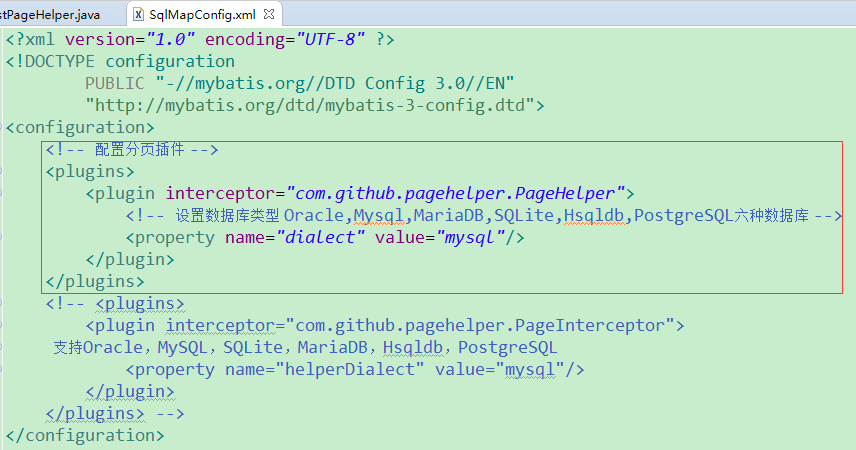
在工程中引入pagehelper插件

## 使用方法

第一步：引入pageHelper的jar包。

第二步：需要在SqlMapConfig.xml中配置插件。

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <configuration>  <!-- 配置分页插件 -->  <plugins>  <plugin interceptor=*"com.github.pagehelper.PageHelper"*>  <!-- 设置数据库类型 Oracle,Mysql,MariaDB,SQLite,Hsqldb,PostgreSQL六种数据库-->  <property name=*"dialect"* value=*"mysql"*/>  </plugin>  </plugins>  </configuration> |



第三步：在查询的sql语句执行之前，添加一行代码：

PageHelper.startPage(1, 10);

第一个参数是page，要显示第几页。

第二个参数是rows，没页显示的记录数。

第四步：取查询结果的总数量。

创建一个PageInfo类的对象，从对象中取分页信息。

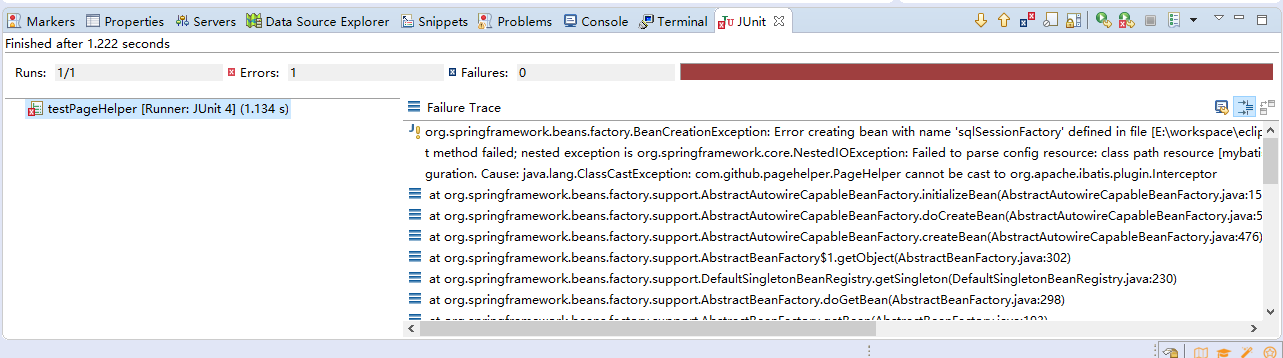
## 分页测试

在src/test/java下添加一个类TestPageHelper进行测试

|  |
| --- |
| **public** **class** TestPageHelper {  @Test  **public** **void** testPageHelper() {  //创建一个spring容器  ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("classpath:spring/applicationContext-\*.xml");  // ApplicationContext applicationContext = new ClassPathXmlApplicationContext("classpath:spring/applicationContext-\*.xml");  //从spring容器中获得Mapper的代理对象  TbItemMapper mapper = applicationContext.getBean(TbItemMapper.**class**);  //执行查询，并分页  TbItemExample example = **new** TbItemExample();  //分页处理  PageHelper.*startPage*(2, 10);  List<TbItem> list = mapper.selectByExample(example);  //取商品列表  **for** (TbItem tbItem : list) {  System.***out***.println(tbItem.getTitle());  }  //取分页信息  PageInfo<TbItem> pageInfo = **new** PageInfo<>(list);  **long** total = pageInfo.getTotal();  System.***out***.println("共有商品："+ total);  }  } |

选择相应的方法，右键测试Run As：Junit Test，如果出现如下问题：

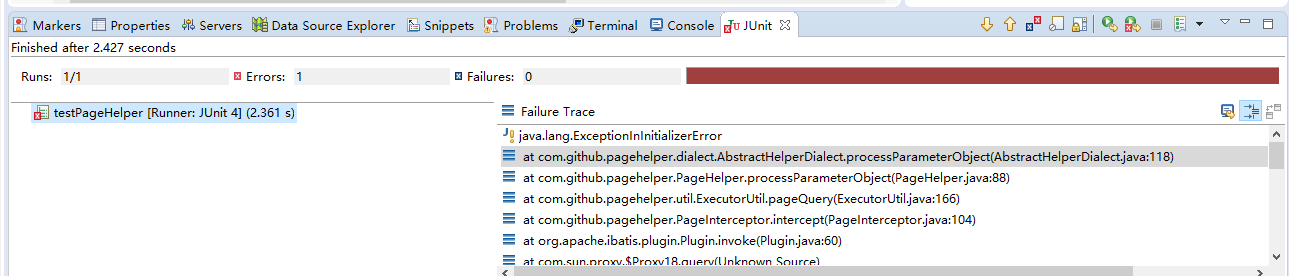
Cause: java.lang.ClassCastException: com.github.pagehelper.PageHelper cannot be cast to org.apache.ibatis.plugin.Interceptor



参考网上资料：PageHelper本质上是一个mybatis拦截器，因此如果要使用，必须在xml中进行配置。

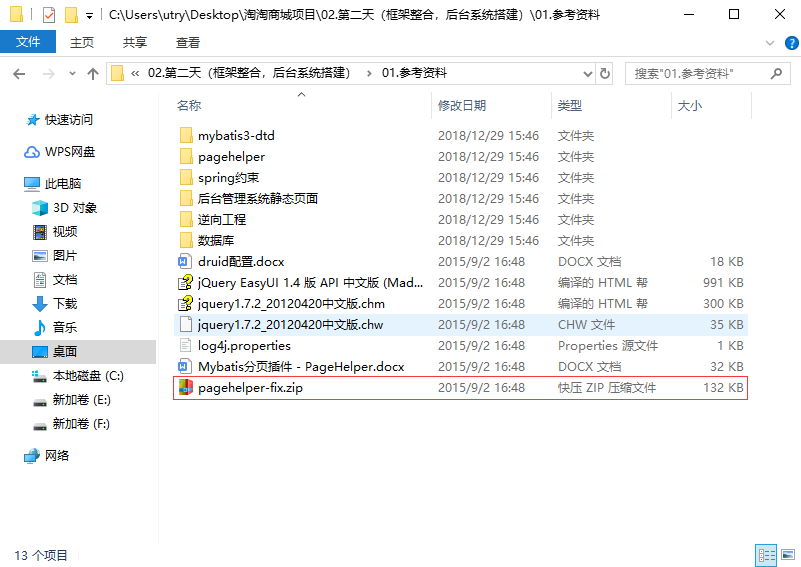


**此处注意：分页插件对逆向工程生成的代码支持不好，不能对有查询条件的查询分页。会抛异常（所以参考源码中实际上使用的修改过的版本，一开始没有考虑之前搭建项目没有用提供的maven仓库而修改了pagehelper版本导致现在访问出错！）**



参考了许多资料，始终会出现各种各样的问题，原因是之前搭建环境的时候pagehelper是由另外提供的，已经放在maven仓库中，因此搭建环境可直接引用而不报错，但在搭建环境的时候没有使用提供的maven仓库，因此修改了pom引用配置，使用了默认的pagehelper，所以导致上述操作出错，因此此处考虑将相关的插件导入本地仓库，修改相关的pom文件进行测试

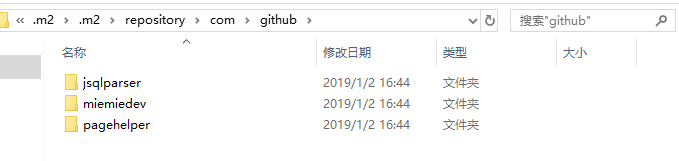
参考链接：<https://blog.csdn.net/yerenyuan_pku/article/details/72774381>



解决方案：

将提供的maven仓库下的com.github下的相关内容替换掉当前使用仓库对应目录下的数据，随后修改taotao-parent的pom文件，将原有pagehelper的版本修改为如下所示，随后重新通过mvn clean install指令更新项目，将taotao-parent重新导入到本地仓库，再次根据文档要求重新进行测试，测试成功！





测试结果显示如下：

